2017 Fifth International Conference on Advanced Cloud and Big Data

Data Deduplication based on Hadoop

Dongzhan Zhang*, Chengfa Liao*, Wenjing Yan*, Ran Tao* and Wei Zheng*
*Department of Computer Science
School of Information Science and Engineering
Xiamen University, China
Email:zhengw@xmu.edu.cn

***Abstract*—Efficient and scalable deduplication techniques are required to serve the need of removing duplicated data in big data processing platforms such as Hadoop. In this paper, an integrated deduplication approach is proposed by taking the features of Hadoop into acount and leveraging parallelism based on MapReduce and HBase so as to speed up the deduplication procedure. In our proposed approach, a new small-file aggregation scheme is proposed, and the new standard of Secure Hash Algorithm-3, Keccak is employed. The effectiveness of our proposed approach is demonstrated by extensive experiments on both artificial datasets with specific characteristics, as well as real world datasets.**

I. Introduction

Nowadays, the amount of data collected from a great variety of sources, which have already been massive, are still growing at an explosive rate. This urges current file systems to provide effective but also efficient services for data storage and retrieval. However, one of the critical challenges faced by file service providers is that they often have to contain duplicate copies of file contents as data are being accumulated. Typical real-world examples can be easily found in increasingly popular Internet social applications, such as Facebook, Youtube and Twitter etc., where file duplication occurs with a high frequency when users share and synchronize files between each other. As unfavourable consequences, the limited storage space is redundantly consumed and the network bandwidth is wasted.

There has been a report [1] indicating that more than 80% of enterprises are seeking deduplication technologies to reduce cost and increase the storage efficiency. Deduplication is the procedure that detects and remove redundants data for a storage system. As depicted in Fig. 1, a deduplication procedure normally consists of five steps which are Chunking, Compute, Lookup, Delete, and Store, which are briefly described as follows.

- ‘Chunking’ is to divide files or data stream into chunks based on which the duplication will be detected and processed.
- ‘Compute’ means computing eigenvalue of each chunk which is will be used in the next step to determine whether duplication occurs.
- ‘Indexing’ is the step to compare the eigenvalue of a chunk with existed ones and add hash to the index if the chunk is new.
- ‘Delete’ is to somehow discard the redundant content of a chunk in case it is found duplicated.

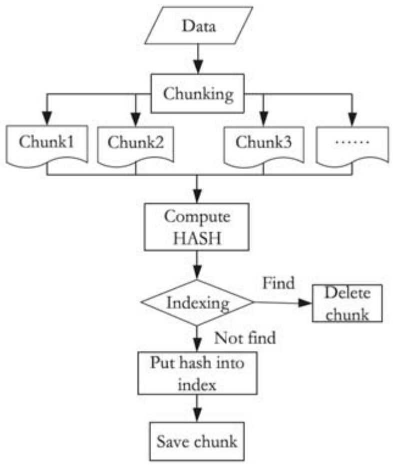

Fig. 1. Procedure of Data Deduplication [2]

- ‘Save’ is to store unique chunks after deduplication.

As the Big Data Era arrives, deduplication plays an increasingly significant role in a storage system. Nevertheless, research efforts are required to address new challenges arising from various aspects such as wise chunk division, performance optimization, data reliability and system scalability.

The main focus of this paper falls in the performance optimization of deduplication. In our study, the role played by Hadoop is twofolded. On one hand, we study data deduplication in the context of Hadoop. As an open-source cloud computing framework, Hadoop has been widely used for massive data storage and processing. However, to the best of our knowledge, few data deduplication technologies have ever been considered to reduce data duplication for Hadoop Distributed File System (HDFS). This motivates our study. On the other hand, our approach tries to exploit parallelism within the above phases of deduplication, which can be implemented with HBase and MapReduce, two key components of Hadoop.

This paper proposes and implements a novel deduplication approach for Hadoop. The main contributions include:

- A new small-file aggregation scheme named PHAF (Parallel Hadoop Aggregation File) is proposed and implemented to improve the performance when Hadoop deals with numerous small files;
- Keccak, which has newly become the SHA-3 (Secure Hash Algorithm 3) standard, is employed to compute

the hash value for each chunk and is evaluated with comparison to its predecessor, the SHA-2 scheme;

- A new fixed-size block, client-based inline deduplication approach which suits the features of Hadoop (as will be discussed later), is proposed and evaluated in a real Hadoop cluster.

The remainder of this paper is organized as follows: in Section II, we give a brief overview of related works. In Section III, we present the proposed data deduplication approach for Hadoop with emphasis on a parallelized small file aggregation scheme and the employed Keccak implemented based on MapReduce. In Section IV, the effectiveness and efficiency of the proposed approach is evaluated. Finally, we conclude the paper in Section V.

II. Related Work

Deduplication, also known as entity resolution, has been widely studied and extensively covered by many surveys, e.g. [2], [3], [4], [5], [6], [7]. Existing deduplication techniques can normally be classified in light of three different facets: (I) chunking granularity, (II) when the deduplication is executed and (III) where the deduplication takes place.

In terms of chunking granularity, the chunking unit of a deduplication technique might be a whole file [8], [9], [10], [11], [5], a fixed size block [12], [13], [14], a variable size block [15], [16], [17], [18], [19], an object [20], [21], [22] or even a byte or bit [23], [24]. In general, the smaller the granularity applied, the higher spatial and time cost will be paid, while the more likely redundant data may be detected and removed. Variable size block deduplication can often achieve the best result on removing data redundancies for its flexibility on chunking size, but it pays the most expensive overhead which might be unacceptable when there is a real-time requirement for data processing. File deduplication is the easiest one to implement and has the least index overhead, but fails to detect the redundancy inside a file. Fix size block deduplication can handle data redundancy between files and inside a file, but as one can easily imagine, it is too sensitive to the insertion or deletion within a file, especially at the beginning of a file.

The time when deduplication is executed can be classified into two patterns, in-line or off-line. The former performs deduplication before data are stored onto disks. In contrast, the latter only executes deduplication after data are stored. In-line deduplication examples include [5], [25], [26], [27], [28]. And Off-line deduplication examples can be found in [17], [29], [30]. Obviously, in-line deduplication can avoid extra storage for redundant data but will introduce latency on data writing. In the converse, off-line deduplication does not affect data writing but may result in unnecessary store and transmit operation for redundant data. Adopting in-line or off-line deduplication should depend on the specific scenario and a good tradeoff between space saving and data processing time should be considered.

When an end-to-end system is considered, the deduplication might take place at the client end, which is the source of data, or the server end, where the data are stored. Correspondingly there are client-based deduplication [31], [32] and server-based deduplication [33], [34]. The client-based deduplication checks the uniqueness of data in local index or in remote index through an agent and performs deduplication before data are delivered to server. The server-based deduplication usually finds more significant redundancies but incurs excessive redundant data traffic over the network.

Recently, there have been a few research attempts such as [35], [36] that consider speeding up data deduplication by distributed processing. These works focus on providing generic distributed deduplication frameworks. In contrast, this paper concentrates on data deduplication in the context of Hadoop. The Dedu deduplication approach presented in [5] is based on HDFS and HBase. However, the chunking granularity and hashing of Dedu diverse significantly with our proposed method.

III. The Proposed Deduplication Approach

In this paper, we take the features of Hadoop into account and propose a fixed size block, client-based in-line distributed deduplication approach for Hadoop. We firstly explain our consideration when shaping this deduplication approach, and then present our design and implementation in details.

A. Our Consideration

The reason why we choose the fixed size block chunking in our implementation of deduplication is based on the fact that the Hadoop Distributed File System (HDFS), which is regarded as the server in our deduplication study, is designed for storing files with streaming data access patterns. In this case, the main drawback of the fixed size block chunking, as mentioned in Section. II is eliminated. This indicates that the fixed size block chunking is particularly suitable for data deduplication in Hadoop.

As HDFS is a file system designed for storing very large files, which means files that are hundreds of megabytes, gigabytes or terabytes in size, so it is particularly important to save the redundant data transmitted over the network . Therefore, client-based in-line deduplication should be a wise choice. Moreover, note that HDFS is natively not good at low-latency data access, so the extra overhead introduced by client-based in-line deduplication may not be regarded as a serious issue for Hadoop. Again, because HDFS is designed for storing large files and the file size now exhibits a trend of rapid growth, the workload of deduplication may exceeds any single node's capability. In order to address this issue, distributed deduplication is adopted to realize the scalability.

B. Design and Implementation

The above-mentioned consideration forms the design of our deduplication framework for Hadoop. An overview of our design is shown in Fig. 2, where the HDFS at the left bottom is regarded as the server where the data will finally be stored and the client-based in-line deduplication is performed in the agent with MapReduce, HBase and possibly HDFS for temporary file storage, as shown on the right.

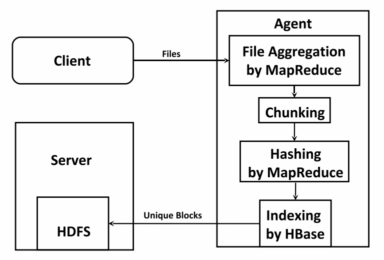

Fig. 2. Structure of Deduplication based on Hadoop

1) File Aggregation: We consider file aggregation as a necessary module for data deduplication in Hadoop, since Hadoop by its current design (version 1.x) is not efficient to deal with a large number of small files. On one hand, since each file consumes certain size of memory from the NameNode, too many files will result in too much memory requirement that the NameNode cannot afford. On the other hand, it is very complicated and error prone to apply our deduplication with MapRedure to many small files.

One way to assemble small files into a large Hadoop file is using the tar-to-seq tool described in [37], which requires firstly compressing all the small files into a tar file and then converting the tar file to a Hadoop sequential file. Apparently, this tar-to-seq tool can only be executed in a single node and is lack of scalability. Therefore, we propose Parallel Hadoop Aggregation File (PHAF) to speed up the file aggregation by parallel processing based on MapReduce.

The procedure of PHAF is described as follows:

1) A list of small files are uploaded;
2) The Map function is initialized with the key parameter set to the file name and the value set to the file content;
3) The output is formatted by HAFie which extends the Hadoop SequenceFile;
4) The MapReduce job is run and the aggregation result is collected.

We will show and discuss PHAF's advantages over tar-to-seq in Section IV.

2) Chunking: After the large sequential file is generated, a fixed size partitioning (FSP) function is employed to chunk the file into equal-size chunks. As most of existing chunking schemes adopts chunk sizes between 4k and 20k bytes. In our implementation, we set the chunk size to 4k bytes. For those files which can not be perfectly divided into 4k-byte chunks, we complement 0-value bytes at the end of the last chunk. The divided chunks, as the output of the FSP function, will be used as the input for hash computation and the unit of data stored to HDFS if being judged unique.

3) Hash Computation: The comparison between new and existing chunks are commonly based on hash computation, for which there are normally three requirements including security, efficiency and anti-collision. However, these requirements conflict with each other. Traditional hash algorithms such as MD5 and SHA-1 (Secure Hash Algorithm) are good at efficiency but poor at security and anti-collision. Nowadays, SHA-2 is widely used for data deduplication. In our approach, Keccak [38], which has recently become the SHA-3 standard, is employed rather than SHA-2.

Keccak is a family of hash functions based on the sponge construction. To compute a hash, the state which can be considered to be a array of bits is initialized by 0. The input is divided into r-bit blocks and a padding function which adds complementary bits to the initial input may be needed if the input cannot be evenly divided. Each block is "absorbed" into the state by the XOR operation and then applying the block permutation, which is a permutation that uses XOR, AND and NOT operations and designed for easy implementation in both software and hardware. After the final block permutation, the leading n bits of the state are "squeezed" out as the desired hash.

Although SHA-3 is not meant to replace SHA-2, as no significant attach on SHA-2 has been demonstrated so far, it would be interesting to compare their efficiency. We compared the hash computation time of Keccak with SHA-2 and found the former outperforms the latter on efficiency. The evaluation details will be presented in Section IV.

4) Deduplication with MapReduce and HBase: HBase, as a distributed database which can provide real-time read/write random-access to very large datasets, is employed in our deduplication approach for storage and query of the computed hash values of chunks. Moreover, the chunk information of the deduplicated files is also maintained in HBase for future file access. The reason of using HBase instead of a relational database system is obviously due to scalability issues. In our implementation, a table named FingerRepo is designed with row keys which can be either chunk hashes or file names, and a column family named chunkFamily. In the case that the row key is a hash, the corresponding chunkFamily contains chunk information such as chunk id, size of chunk, number of references, number of associated files, name of the associated file, offset within the associated file and block address. In the case that the row key is a file name, the corresponding chunkFamily contains chunk list.

As a summary, the overall deduplication procedure can be described as follows: (I) input files are aggregated by PHAF; (II) the generated file is divided into chunks with size of 4k bytes; (III) Keccak is used to compute hash for each chunk and the computed hash is checked in the FingerRepo; (IV) if the computed hash does not exist in FingerRepo, the chunk is stored into HDFS and the FingerRepo is updated accordingly; otherwise, the chunk is not stored and the pointer to the stored single copy is returned to update the file's chunk list for future file access. Apparently, one can easily implement these operations, such as fixed size partition, Keccak hash computation and hash check, by MapReduce so as to acquire acceleration for the whole deduplication process.

5) File Access after Deduplication: Once deduplication is applied to the input files, the files that contain duplicated chunks will share a single copy. As mentioned in the previous subsection, the access information about this copy is maintained in the FingerRepo table based on HBase. Therefore, in order to access a file stored in the server (i.e., HDFS), the

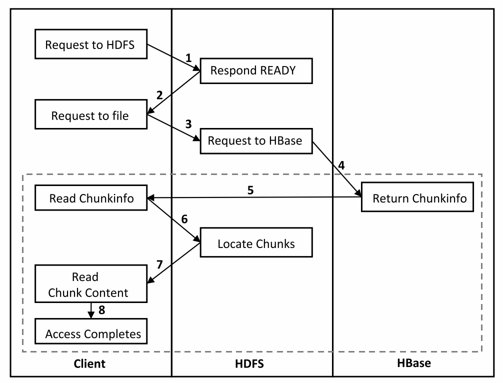

Fig. 3. Steps of File Access after Deduplication

client needs to firstly query to the HBase before retrieving file contents from the HDFS. The detailed file access procedure is shown in Fig. 3.

IV. Evaluation

In this section, the experimental environment we used is firstly introduced. Next, the PHAF that is used to aggregate small files into a large one is compared to the tar-to-seq tool. Then, we evaluate the execution time of two hash computation algorithms, namely Keccak and SHA-2, respectively. Finally, the deduplication approach as a whole is evaluated on three artificial datasets and its efficiency and effectiveness are discussed based on the results.

A. Experimental Setting

The Hadoop platform that acts as the testbed in our experiment is built on top of 4 distributed heterogeneous nodes which connects to each other within a LAN. The master node has Intel Core i5 CPU@2.3GHz, 4GB memory and 320GB hard disk, and the other three slave nodes have Intel Core i3 CPU@3.5 GHz, 1GB memory and 200GB hard disk. The operating system running in the master node is MAC OS X Yosemite x64 and the slave Ubuntu14.04 x64. For HBase, the HMaster is deployed in the master node and HRegionServer and HQuorumPeer are deployed to each node. The version of Hadoop used is v1.1.2 and the JDK used is version 1.7.

B. Comparison between PHAF and Tar-to-Seq

In the first experiment, a dataset consisting of different figures is used as input. The size of each figure is between 100KB and 5MB. There are 300 figures in total, which results in a total size of 600MB. As we use the default setting of data block size of Hadoop (64MB), our test files are obviously small files.

The proposed PHAF approach is implemented and evaluated in the Hadoop platform as specified in the previous subsection. The tar-to-seq tool is downloaded from [37] and tested in the master node and the slave node respectively. Note that the execution time of tar-to-seq tool shall include the time used to compress all small files into a tar file.

TABLE I. Time Cost of PHAF and Tar-to-Seq

Approach	Running Environment	Files (total size/number)	Time Cost(sec.)
PHAF	4-node Hadoop	(600MB/300)	57
tar-to-seq	Single Node, Master	(600MB/300)	65
tar-to-seq	Single Node, Slave	(600MB/300)	85

Table I shows the time cost results for PHAF and tar-to-seq respectively. Using the same files as input, the PHAF which runs on the 4-node Hadoop platform takes less time than the tar-to-seq tool which runs either on the single master node or the single slave node to complete the file aggregation. Although the time cost difference between the PHAF and the tar-to-seq on the master node is not very significant, it can be easily imagined that the time cost of PHAF can be potentially reduced by adding more slave nodes and a more significant time improvement can be made. When there are too many files to process, the tar-to-seq tool's performance may sharply downgrades due to performance bottleneck arising from memory or I/O in a single node.

C. Comparison between Keccak and SHA-2

In the second experiment, we compare the efficiency of Keccak and SHA-2. To be fair, we specify the output size of both algorithms to be 224, which is sufficient to meet the security and anti-collision requirements. The Keccak and the SHA-2 algorithm are both implemented by JDK 1.7 and tested in the master node. In our evaluation, three inputs are used. The first is a text file with a size of 4KB. The second is a JPG file with a size of 40KB. And the third is the source code of Git, a compressed file with a size of about 4MB. The Keccak and the SHA-2 are applied to these inputs respectively and the average time cost is computed over 100 measurements.

TABLE II. Time Cost of Keccak and SHA-2

Algorithm	TimeCost for Input1	TimeCost for Input2	TimeCost for Input3
Keccak	0.63ms	3.23ms	41ms
SHA-2	5.18ms	9.02ms	67ms

The detailed time cost of applying the Keccak and the SHA-2 to the three inputs are presented in Table II. One can easily see that the Keccak outperforms the SHA-2 on efficiency for all cases of inputs, especially when the input is small. This indicates the Keccak shall be a better choice than the SHA-2 for hash computation for the deduplication. Moreover, as the Keccak is designed for easy implementation by hardware, by so the Keccak can still meet the efficiency requirement even if there are enormous chunks to be hashed.

D. Overall Evaluation

In the last experiment, we examine the effectiveness and efficiency of the whole deduplication approach which employs PHAF for file aggregation and Keccak for hash computation. The deduplication approach takes a bunch of files as input. These files are firstly aggregated by PHAF into a Hadoop sequential file. Then the sequential file is evenly divided into chunks by fixed size partition. For each chunk, the hash value is computed by Keccak and checked in the FingerRepo based on HBase. And then different further operations are performed

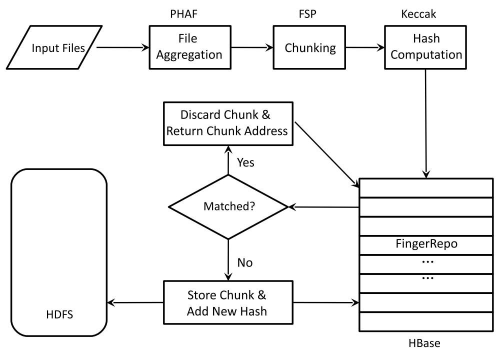

Fig. 4. Flowchart of the Deduplication

according to whether the hash value has existed in the FingerRepo. The entire deduplication procedure is demonstrated in Fig. 4.

Two metrics are considered in our evaluation. The first is processing time, which means the duration between when the input files are received and when the operations following deduplication completes. We use T_{total} to denote the processing time. Then we have:

$$T_{total} = T_{pre} + T_{core} + T_{storage} + T_{other} \tag{1}$$

where T_{pre} means the time needed for file aggregation, T_{core} means the time spent on chunking and hash computation and checking, $T_{storage}$ is the time taken for storage operation, and T_{other} includes the time cost on changing the MetaData of HDFS, updating the HBase records and network delay etc. The other metric we are concerned is Deduplication Rate, which is denoted by DR and computed by

$$DR = \frac{V_{initial} - V_{storage}}{V_{initial}} \tag{2}$$

where $V_{initial}$ is the total size of the initial files and $V_{storage}$ is the total size of the stored chunks after deduplication.

The datasets we used in this evaluation derive from figures and log files of a MOOC website. We make some files have extra copies in the dataset so that a certain degree of data duplication can be guaranteed. After processing, we use three datasets of which the total size, total number of files and the estimated duplication percentage are presented in Table III. Note that the actual duplication percentage of each dataset might be higher than the value presented in Table III.

TABLE III. CHARACTERISTICS OF THE USED DATASETS

Attributes	Dataset 1	Dataset 2	Dataset 3
Total Size(MB)	600	600	1000
Number of Files	10	300	300
Duplication Percentage	50%	50%	10%

For each dataset, the evaluation is carried out twice and each time corresponds to an extreme case. In the first time, the HDFS and the FingerRepo are empty. This means all duplicated data are inside the datasets. In the second time, as all file contents have already been stored in the last evaluation, all inputs can be regarded as duplicated data. With these settings, we evaluate our deduplication approach and the deduplication rate (DR) and processing time for each evaluation are measured and shown respectively in Table IV and Table V. Note that Dataset1 and Dataset2 have the same amount of data but different number of files, while Dataset2 and Dataset3 have the same number of files but different amount of data.

TABLE IV. RESULTS FOR THE FIRST EVALUATION

Metrics	Dataset 1	Dataset 2	Dataset 3
Processing Time(s)	2694	4193	4100
Size of Stored Data(MB)	306	322	1050
Duplication Rate	49%	45%	-5%

TABLE V. RESULTS FOR THE SECOND EVALUATION

Metrics	Dataset 1	Dataset 2	Dataset 3
Processing Time(s)	516	592	1136
Size of Stored Data(MB)	0	0	0
Duplication Rate	100%	100%	100%

From the observed results, we list key findings as follows:

- With the dataset size and the duplication percentage fixed, the processing time of deduplication grows and the DR drops as the number of files increases. For example, in Table IV, the processing time of dataset2 is longer than that of dataset1 while the DR is slightly lower. With respect to the processing time, it is clear that more input files result in more key/value pairs to process during the PHAF phase and requiring more time to query from FingerRepo. With regard to the DR, more input files may render to more padding operations during the PHAF and the FSP phases. As a result, the amount of stored data may increase. This leads to decrease of DR. However, note that even though the number of files contained in dataset2 is as many as 30 times of dataset1, the difference of processing time shown in Table V is not too significant. This implies the necessity of applying the PHAF in our deduplication approach.
- With the number of files fixed, the processing time of deduplication depends on both the total size of data and the duplication percentage. For example, in Table IV dataset3, which has more data but lower duplication percentage in comparison to dataset2, takes less processing time than dataset2. The results of dataset2 and dataset3 in Table V shows that when they have the same duplication percentage, the processing time is roughly in proportion to their data amount.
- There seems to be a strange result in Table IV where the DR result of dataset3 is negative. This is because the duplication percentage of dataset3 is only 10% and the padding operations carried out during PHAF and FSP may add more complementary bytes to the stored data than the data that is deduplicated.

We finally apply our deduplication approach to a bunch of real data from a MOOC website, which consists of 500 users' files with a total size of 15GB. After deduplication through our approach, the stored data in HDFS turns out to be 11.3GB. Namely, about 25% percent of storage space is saved. This indicates our approach is promising for Hadoop platforms in practice.

V. Conclusion and Future Works

This paper focuses on the issues arising from duplicated data within Hadoop platforms and proposes an integrated deduplication approach with the aid of Hadoop. In the proposed approach, a new file aggregation scheme based on MapReduce is proposed and the recent SHA-3 standard Keccak is used for hash computation. Moreover, the overall deduplication procedure is implemented based on MapReduce and HBase so as to provide scalability to cater to the challenges brought by the Bigdata Era. Experimental results suggest that our design of deduplication framework is reasonable and the proposed approach as a whole is efficient and effective in practice. Future works may include further optimization on the chunking algorithm and the I/O operations, or migrating the deduplication framework to Spark platforms for better execution performance.

Acknowledgment

The work is supported by National Natural Science Foundation of China (NSFC, Grant No. 61671397, 61672439).

References

[1] R.-K. Sheu, S.-M. Yuan, W.-T. Lo, and C.-I. Ku, “Design and implementation of file deduplication framework on hdfs,” *International Journal of Distributed Sensor Networks*, vol. 10, no. 4, p. 561340, 2014.

[2] X. Zhang and M. Deng, *An Overview on Data Deduplication Techniques*. Cham: Springer International Publishing, 2017, pp. 359–369.

[3] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, “Duplicate record detection: A survey,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 1, pp. 1–16, Jan 2007.

[4] L. Getoor and A. Machanavajjhala, “Entity resolution: Theory, practice & open challenges,” *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 2018–2019, Aug. 2012. [Online]. Available: http://dx.doi.org/10.14778/2367502.2367564

[5] Z. Sun, J. Shen, and J. Yong, “A novel approach to data deduplication over the engineering-oriented cloud systems,” *Integrated Computer-Aided Engineering*, vol. 20, no. 1, pp. 45–57, 2013.

[6] I. F. Ilyas and X. Chu, “Trends in cleaning relational data: Consistency and deduplication,” *Foundations and Trends in Databases*, vol. 5, no. 4, pp. 281–393, 2015. [Online]. Available: http://dx.doi.org/10.1561/1900000045

[7] Y. Shin, D. Koo, and J. Hur, “A survey of secure data deduplication schemes for cloud storage systems,” *ACM Comput. Surv.*, vol. 49, no. 4, pp. 74:1–74:38, Jan. 2017. [Online]. Available: http://doi.acm.org/10.1145/3017428

[8] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur, “Single instance storage in windows 2000,” in *Proceedings of the 4th USENIX Windows Systems Symposium*. Seattle, WA, 2000, pp. 13–24.

[9] “Centera: Content addresses storage system, data sheet.” http://www.emc.com/collateral/hardware/data-sheet/c931-emc-centera-cas-ds.pdf, accessed April 4, 2017.

[10] “Justcloud,” http://www.justcloud.com/, accessed April 4, 2017.

[11] “Mozy,” http://mozy.com/, accessed April 4, 2017.

[12] S. Quinlan and S. Dorward, “Venti: A new approach to archival storage.” in *FAST*, vol. 2, 2002, pp. 89–101.

[13] “Dropbox,” http://www.dropbox.com, accessed April 4, 2017.

[14] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer *et al.*, “Oceanstore: An architecture for global-scale persistent storage,” *ACM Sigplan Notices*, vol. 35, no. 11, pp. 190–201, 2000.

[15] A. Muthitacharoen, B. Chen, and D. Mazieres, “A low-bandwidth network file system,” in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5. ACM, 2001, pp. 174–187.

[16] W. Dong, F. Douglis, K. Li, R. H. Patterson, S. Reddy, and P. Shilane, “Tradeoffs in scalable data routing for deduplication clusters.” in *FAST*, vol. 11, 2011, pp. 15–29.

[17] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki, “Hydrastor: A scalable secondary storage.” in *FAST*, vol. 9, 2009, pp. 197–210.

[18] W. Xia, H. Jiang, D. Feng, and Y. Hua, “Silo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput.” in *USENIX Annual Technical Conference*, 2011.

[19] B. Zhu, K. Li, and R. H. Patterson, “Avoiding the disk bottleneck in the data domain deduplication file system.” in *Fast*, vol. 8, 2008, pp. 1–14.

[20] D. Kim, S. Song, and B.-Y. Choi, “Safe: Structure-aware file and email deduplication for cloud-based storage systems,” in *Data Deduplication for Data Optimization for Storage and Network Systems*. Springer, 2017, pp. 97–115.

[21] C. Liu, Y. Lu, C. Shi, G. Lu, D. H. Du, and D.-S. Wang, “Admad: Application-driven metadata aware de-duplication archival storage system,” in *Storage Network Architecture and Parallel I/Os, 2008. SNAPI’08. Fifth IEEE International Workshop on*. IEEE, 2008, pp. 29–35.

[22] F. Yan and Y. Tan, “A method of object-based de-duplication.” *JNW*, vol. 6, no. 12, pp. 1705–1712, 2011.

[23] F. Douglis and A. Iyengar, “Application-specific delta-encoding via resemblance detection.” in *USENIX annual technical conference, general track*, 2003, pp. 113–126.

[24] D. Meister and A. Brinkmann, “Multi-level comparison of data deduplication in a backup scenario,” in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM, 2009, p. 8.

[25] A. El-Shimi, R. Kalach, A. Kumar, A. Ottean, J. Li, and S. Sengupta, “Primary data deduplication-large scale study and system design.” in *USENIX Annual Technical Conference*, vol. 2012, 2012, pp. 285–296.

[26] K. Srinivasan, T. Bisson, G. R. Goodson, and K. Voruganti, “idedup: latency-aware, inline data deduplication for primary storage.” in *FAST*, vol. 12, 2012, pp. 1–14.

[27] B. K. Debnath, S. Sengupta, and J. Li, “Chunkstash: Speeding up inline storage deduplication using flash memory.” in *USENIX annual technical conference*, 2010, pp. 1–16.

[28] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble, “Sparse indexing: Large scale, inline deduplication using sampling and locality.” in *Fast*, vol. 9, 2009, pp. 111–123.

[29] “Emc: Achieving storage efficiency through emc celerra data deduplication,” http://china.emc.com/collateral/hardware/white-papers/h6265-achieving-storage-efficiency-celerra-wp.pdf, accessed April 4, 2017.

[30] “Ibm white paper: Ibm storage tank - a distributed storage system,” https://www.usenix.org/legacy/events/fast02/wips/pease.pdf, accessed April 4, 2017.

[31] “Emc: Avamar,” http://www.emc.com/backup-and-recovery/avamar/avamar.htm, accessed April 4, 2017.

[32] “Symantec:puredisk,” http://www.symantec.com/netbackup-puredisk, accessed April 4, 2017.

[33] “Emc: Networker,” http://www.emc.com/domains/legato/index.htm, accessed April 4, 2017.

[34] “Nec: Hydrastor,” https://www.necam.com/hydrastor/, accessed April 4, 2017.

[35] L. Kolb, A. Thor, and E. Rahm, “Dedoop: Efficient deduplication with hadoop,” *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1878–1881, Aug. 2012. [Online]. Available: http://dx.doi.org/10.14778/2367502.2367527

[36] X. Chu, I. F. Ilyas, and P. Koutris, “Distributed data deduplication,” *Proc. VLDB Endow.*, vol. 9, no. 11, pp. 864–875, Jul. 2016. [Online]. Available: http://dx.doi.org/10.14778/2983200.2983203

[37] “A million little files,” https://stuartsierra.com/2008/04/24/a-million-little-files, accessed February 3, 2017.

[38] “The keccak sponge function family,” http://keccak.noekeon.org/specs_summary.html, accessed February 28, 2017.